MILLERS GRAPHICS

JULY 1984

$1.50

THE SMART PROGRAMMER

To start off this issue we have a couple announcements to make about some new products for the 99/4A. We were recently contracted by SCI TECH to work on the firmware for a RAM Disk card (Disk Emulator) for the 99/4A PE Box. SCI TECH is an established computer related company in Southern California that is branching out to manufacture products for the TI 99/4A.

This RAM Disk card is slated to be user expandable from 64K up to 256K in 64K increments. There is also space on the board to plug in an optional 32K for use as memory expansion, if you don't already have it. This card will have an optional external power supply available for it which will allow you to turn off your system without losing the contents of the RAM Disk. The RAM Disk can hold ALL types of files and programs, including Forth, and depending on the size you can have up to 127 different files stored in it at one time. If the card does not have the external power supply attached it will automatically format itself on system power up. You can also format it from the complete Advanced Disk Manager that comes with it. The disk manager allows you to format all or part of the RAM Disk. The unformatted part can be used by Assembly programmers as extra ram space. The RAM Disk looks exactly like a disk drive to the system except that it has a much higher transfer rate, approx 10-15 times faster. The load time for a 50 sector Basic program in single density format was approx 18 sec. With the RAM Disk this same program loaded in 1.5 seconds, very fast!

The RAM Disk can be configured, from a command line or a running program, as DSK1 or DSK5 and it can always be accessed as RDSK. The card will also add some new CALL's to both Basic and X-Basic one of which is:

CALL DIR This command or program statement will allow you generate a paged Catalog of any disk in your system WITHOUT losing your program or data in memory! After the directory is finished your program is still intact and can continue execution.

The RAM Disk will be a nice addition to our systems and a great speed enhancer! I currently do not know the exact price or the release date but we will let you know in the next issue.

Millers Graphics is about to release a new Memory and Disk Diagnostics program for the 99/4A that is compatible with the TI Disk Controller and the CorComp Disk Controller. This program, which is written by Steve Mildon, will do a complete memory check for VDP, Scratch Pad and Memory Expansion RAM and report any errors it may find. It will also do a complete disk diagnostic check which has the following features:

Check and display the disk drive motor speed (RPM), check the bit map for bad sectors and report their numbers and check the disk for fractured files and report their names. It will also seek out any sector on command and read it or write to it with a verify and it will seek out any track on command and read it or write to it with a verify. It will allow you select different head step times to test your drives with to find the optimum time. After running this program there won't be any doubt in your mind as to the status of your systems memory, diskettes or disk drives. At 19.95 plus 1.50 shipping and handling we believe that you will find it to be a nice utility that is well worth the money. (It should be ready to ship in approx 4 weeks.)

Q&A

There have been a number of questions regarding Basic's and Extended Basic's use of SAVE and OLD with Disk and Cassette. These questions have led us to believe that we should explain their use of memory a little better.

BASIC - TI Basic does NOT recognize memory expansion!! When you load a Basic program either with cassette or disk it is loaded into VDP RAM and it stays there. A Basic program loaded through Basic is always in VDP RAM. If you load a Basic program through Extended Basic and you have memory expansion then it is moved out to high memory expansion after it is fully loaded. With this in mind it is useless to purchase memory expansion unless you are going to use Extended Basic, Editor Assembler, Mini Memory or one of the many programs that require it. If you have a Mini Memory module and Expansion Memory you can save and load Basic programs to and from high memory expansion with SAVE EXPMEM2 and OLD EXPMEM2 or you can use this area for files ie; OPEN #1:"EXPMEM2".

EXTENDED BASIC - TI Extended Basic DOES recognize memory expansion and to get the full power of Extended Basic we strongly recommend that you use it with memory expansion. If you are using a cassette based system when you SAVE a program the line number table and crunched program are moved into VDP RAM and then saved to cassette. If the program is too large to fit in VDP RAM the computer returns an ERROR and you can not save it without editing out some code. What this means is that with a cassette system and memory expansion your program size is limited to the size of VDP RAM. However your programs can contain a greater amount of string and numeric variables since the symbol tables for these are generated AFTER you type in RUN. You might be able to get it to save by using more multiple line statements. Each line number adds 6 bytes to your program. 4 bytes for the line number table entry, 1 byte for the line length and 1 byte for the end of line marker (character). Another item to look at is your variable names, each character in the name is one byte. So, if you have a variable name that is 10 bytes long and you use it 20 times in your program you've used up 200 bytes of program space. We've also noticed that shorter variable names, 1-3 characters, run faster. One more item to consider is using single character numeric variables in place of constants such as 1,2,3 etc. or any other number that is quite frequently used in your program. Each direct number in your program requires 2 bytes + the number of digits + the decimal point. So the number 100 uses up 5 bytes and the number 123.45 uses up 8 bytes. If you assign this number to a variable like A=100, this assignment is 7 bytes, 2 for A= and 5 for 100. However, now whenever you use the variable A it only uses 1 byte instead of the 5 bytes that 100 uses. This will shorten up your program length but every variable added to your program slows it down since it has to search through the symbol table each time you reference it. And lastly, as always, you can remove any REM statements since they use up 6 bytes for the line number, 1 byte for the REM token and 1 byte for EVERY character that follows it. Also, if a REM statement is logically in line with your program code it will slow it down a little. These program shorting tips also apply to Basic.

If you have a disk system and memory expansion you are able to write and save large programs that can use most of the 24K high memory expansion. Extended Basic only uses the low memory expansion for Assembly files. Assembly files can also be loaded with an Extended Basic program into high memory expansion provided they do not overlap onto your Extended Basic program. With a disk system if you save a program that is too large to fit in VDP RAM it changes it from a PROGRAM (memory image) type file to a INT/VAR 254 type file and writes out 1 record at a time. When this happens the save and load times are much longer and you will notice that your disk controller light and 32K light alternately blink on and off. If you load a a PROGRAM type file you will notice that the disk controller light blinks rapidly. Then, after the program has finished loading into VDP RAM, the 32K light will blink on when the program is transferred to high memory expansion.

Due to the nature of the TI operating system and the Disk Controller DSR an PROGRAM type file MUST fit into VDP RAM in order to Save or Load it. This is also true of the PROGRAM type files that the PRK and

Adventure modules produce. These files are mainly data but because of the structure of a PROGRAM type file they are very compact and very quick to load. We seem to get a number of questions on this type of file so lets explain it a little better. First off the Diskette doesn't care what is written on it, it will hold whatever is written to it. It is only the DSR and Languages that care what the file type is. The DSR is set up in such a way that ANY file that has a FIXED length can be read as a RANDOM type file. ANY file that is VARIABLE MUST be read as a SEQUENTIAL type file. And, ANY PROGRAM (memory image) type file is loaded into VDP RAM first and then it is moved to its proper location after it is fully loaded. Were it is moved to is determined by the program that loaded it. It may stay in VDP RAM or it may be moved out to memory expansion. The loading program can move pieces of it into different locations, like TI Forth does with the FORTHSAVE file. Part of this file, User Variables, is moved into low mem and the resident vocabulary is moved into high-mem expansion. A memory image file is just that, it is a copy of a block of RAM that has been saved out to disk, there are no control bytes or loader tags mixed in with the code. There are a few bytes in the beginning of files that pass info to the loader. In X-Basics case the PROGRAM file contains the line number table and the crunched program as they reside in memory. In the PRK case this file is a copy of RAM that contained your data. You can not RUN this file but the PRK program can load it into memory and use selected parts of it for data. Forth's BSAVE function is very similar to a PROGRAM type file. If TI Forth used file I/O instead of sector I/O and put file headers on the disk these BSAVEd files would probably be labeled as PROGRAM. I hope this has helped and not made it more confusing than it was.

What happened to the rumors about Toys R Us and/or GE buying the rights to produce the 99/4A?

Last we item that we herd about this was that TI was asking TOO MUCH money for the rights so both companies decided not to buy! Also everything has been quiet on the 99/8. At one time we herd that Control Data Corp. was thinking about buying this or a version of the 99/5, but that was a long time ago.

What is the AVDP chip (TMS 9228) or the new Video Display processor that TI is about to produce and can it be used to replace the TMS 9918A video chip in the 99/4A?

The TMS 9228 is a new development from TI. It is not currently in production, it is still in the testing stages so some of the specifications may change. According to our friend, W.R. Moseid, sample chips should be ready for release in the first quarter of 85' and production quantities should be available around the last quarter of 85'. TI would like to see this chip become a standard for VideoText interface units.

This new chip will address from 16K to 256K of video RAM. It allows up to 10 sprites per row since it has ten sprite registers. It has 5 graphics modes and 2 text modes. Text mode I is 40 columns and Text mode II is 80 columns. Both Text modes allow multicolored text with a special mode setting. The graphics modes go from the standard 32 column Basic mode up to a true bit map mode that has 256 x 210 pixel resolution and requires 26880 bytes of RAM to use it. It is a true bit map mode in that each individual pixel can be set to one of 16 colors. There is an even higher resolution mode that has 512 x 192 pixels know as Graphics IV mode. This mode maps its color like our current bit map mode which is limited to 2 colors per character's horizontal pixel row.

Some of the items built into this chip are as follows: Sprites can be labeled into 8 different groups and there is a read only register that contains the group numbers that have a coincidence. Sprites may also be multicolored, up to two colors per horizontal pixel row. The color palette allows you to use 16 colors of a 512 color palette at any one time. This palette is controlled by VDP register numbers 32-63 so you have a lot of program control. There is a hardware horizontal scroll that allows you to scroll from 1 to 256 pixels at a time. It can be used in any of the Graphics modes but not in the Text modes. There is also a vertical scroll that scrolls from 1 to 192 or 210 pixels and it can be used in all modes. The chip also has a block move command that can be executed on a programmable interrupt and a sound generator built into it like the 99/4A's.

A few of the other nice items about this chip are that it allows the CPU to directly access the VDP RAM and it allows a programmable interrupt on a prespecified horizontal scan line. This would allow you to use the block move when the scan got to the middle of the screen. With this you could move a new sprite attribute and motion table into place or you could change the VDP registers for these tables and easily generate up to 64 sprites on the screen. Or, you could use the scrolling registers to scroll different portions of the screen in different directions depending on where the scan was. Once again TI has been able to pack a lot of power in a small package.

Now that you have the good news here comes the bad news. NO you can not just plug this chip into your console. In order to use this chip you would have to modify both the VDP RAM and all of the circuit traces to the AVDP chip. This chip is available in a 40 pin package similar to the 9918A but all of the pin outs are different. Also our consoles currently contain 4116 RAM chips for VDP and the AVDP chip can only use 4416 or 4164 RAM chips. TI's 4416's and 4164's also have different pin outs than the 4116's but another manufactures may be pin compatible. There may also be a problem with TI Basic, in some of the Basic GROM code TI loads VDP Register 4 with >8. This places the pattern descriptor table at >4000. In a 16K system this is the same as >0000 since >3FFF=16K. With more than 16K installed the pattern descriptor table will be in the wrong place for most of Basic's character related statements. Until someone puts one of these chips in a console we won't know for sure. This chip also requires a 35 MHz clock with some added circuitry, the VDP clock in the 4A is 10.7 MHz. If anyone is successful in installing this chip in a console please lets us know, we would like to publish the details.

While we are talking about VDP chips let me add that the 9918A CAN NOT be directly hooked up to a Red, Green, Blue (RGB) monitor. You can only hook it to a composite video input (video in) jack. One of the Users Groups in Pennsylvania is going to try and replace the 9918A with a 9928A which does drive a RGB monitor. We'll keep you posted on the results and the details.

A Little something extra for Christmas ... but only if you've been REAL good!

We recently had the opportunity to learn about a new computer system that TI is about to release. It is named "The Explorer". This system was designed for High-Performance Symbolic Processing, to support the capabilities of LISP and the systems powerful software development system. According to TI this set up is ideal for developing software that uses Artificial Intelligence.

The Explorer computer system comes in three units. The System Unit, the system Console and the Mass storage unit. The System unit contains the 32 bit LISP processor running at 7 MHz, 16K x 56 bits of what we call scratch pad RAM and 2 megabytes of main memory RAM (standard) expandable to 16 megabytes per system. It also contains a 32 bit data and address bus called NuBus and can address a virtual memory space of 128 megabytes. This NuBus (developed by TI) has a transfer rate of 37.5 megabytes per second. There is also an Eithernet LAN Interface, Parallel and RS232 ports, a graphics control logic with 1 megabit bit map using 120 nanosecond RAM and the interface controller for the system console. This interface hooks the monitor, keyboard, mouse, microphone and headset to the System unit via a fiber optics cable. This allows for a 68 megapixel per second transfer rate! The 17 inch monitor has a resolution of 1024 x 808 pixels. The keyboard is similar to the TI Pro's except they have added more keys for a total of 111. The mouse is an optical 3 button unit with 200 dots per inch resolution. The Mass storage unit(s) (up to four per system) house two 5 1/4 devices (ie: hard disk and tape drive). By placing all hard disks in these units it is possible to have up to 896 megabytes of formatted hard disk space with a minimum of 1 megabyte per second transfer rate. The system also comes with a TI 855 printer. All this for only $52,500 to $66,500..... Lets see, the 4A started at $1,400 and came down to ... Hmmmmm

Well.. Dream On... Christmas is just around the corner!

By the way, if you would like a brochure to put under your pillow try writing to:

Texas Instruments
P.O. Box 809063
Dallas, TX, 75380-9063

OOPS!

Well we spoke a little too soon about the May issue. Right after the May issue went to press we received the following info from Heiner Martin, of West Germany. It appears that the GPL Link program we published has a bug in it. If you are using the routine within an assembly language program that executes a BEEP or HONK sound and then execute another GPL BEEP or HONK routine your computer will lock up. The BEEP and HONK sounds are interrupt driven and as such they modify R11 of the GPL Work Space if you access a sound list in GROM. If you are only using the routine with the CALL EXEGPL then it returns back to X-Basic and everything is properly restored. So, the CALL LOAD version we published is OK since you return to X-Basic after every GPL Link. Listed below is the version that Heiner Martin sent us. This one works much better with your assembly language programs.

-- Thank you Mr. Martin --

```
UTLWS  EQU  >2038
SUBST  EQU  >8373
GRMRA  EQU  >9802
GPWS   EQU  >83E0

GPLLNK DATA UTLWS
       DATA GPLLN1

GPLLN1 MOVB @GRMRA,R0
       SWPB R0
       MOVB @GRMRA,R0
       SWPB R0
       AI   R0,-3
       MOVB @SUBST,R1
       SRL  R1,8
       AI   R1,>8300
       INCT R1
       MOV  R0,*R1
       SWPB R1
       MOVB R1,@SUBST
       LI   R3,>2000
       MOV  *R3,R2
       LI   R0,GPLLN2
       MOV  R0,*R3
       MOV  *R14+,@>83EC
       LWPI GPWS
       B    @>0060

GPLLN2 LWPI UTLWS
       MOV  R2,*R3
       RTWP
       END
```

Mr. Martin also sent us the following information on our GROM Chip 0 Memory Map.

We stated that we thought the area from >16DC to >17FF was not used, well it appears that we should have done a little more digging. Mr. Martin has informed us that this area contains the ASCII code tables that are used by the SCAN routine. Here is the list that he sent to us:

16E0 - 16FF Joystick Codes
1700 - 172F Small Characters
1730 - 175F Shift Table
1760 - 178F FCTN Table
1790 - 17BF CTRL Table
17C0 - 17EF Table for modes 1 & 2

For everyone with a CorComp Double Density Disk Controller card Tom Knight sent us the following program. This short program allows you to load the CorComp Disk Manager from the Editor Assembler module with option 3 - Load and Run. Thanks Tom.

```
       IDT  'LOADMNGR'
       AORG >2700
       DEF  MGR

MGR    LWPI >83E0
       MOV  R11,@>8300
       LI   R12,>1100
       SBO  >0000
       SBZ  >000B
       BL   @>44F2
       NOP
       SBZ  >0000
       MOV  @>8300,R11
       B    *R11
       END
```

PEEKING AROUND

Our GROM maps for this issue became a little more involved, and time consuming, then we planned on. There was so much information that we wanted to pass on to you they they ended up a little long. For this reason we took out the PC column this issue and, as you may have already noticed, there wasn't much room left for the programs we wanted to run. Next issue we will lighten up a little on the maps and devote more space to programs. These GROM Maps complete the memory mapping of console GROM, so we will move to another area next issue.

CONSOLE GROM CHIP 1

>2000	**GROM HEADER**
>2000	>AA Valid GROM Header Identification Code
>2001	>02 Version number
>2002	>01 Number of Programs.
>2003	>00 Reserved
>2004	>0000 Address of Power Up Header .... none here
>2006	>214D Address of Application Program Header
>2008	>0000 Address of DSR Routine Header
>200A	>4D1A Address of Subprogram Header (in GROM Chip 2)
>200C	>0000 Address of Interrupt Link .... none in GROM
>200E	>0000 Reserved for future? expansion.
	GROM CHIP 1 VECTOR TABLE (>2000 offset)
>2010	>4417 Routine to begin execution of Basic program in GROM
>2012	>4195 Routine to clear flags & set up keyboard
>2014	>460B Routine to parse (scan) an inputted command line
>2016	>466C Routine to generate the SYNTAX ERROR message.
>2018	>467E Routine to restore cursor position after Error
>201A	>4192 Secondary entry point for Basic Interpreter
>201C	>47F1 Routine that CALLs routines in GROM 0 to load characters
>201E	>436D Routine to move blocks of VDP RAM
>2020	>46AB Routine to reset the length byte for strings and numerics
	ERROR MESSAGES DATA TABLE
>2022	The Error messages in this table have a >60 (96) offset added to them for Basic so they are not readable with the Debugger's M G2022 214D command. (1st byte=length - Next bytes=message)
	APPLICATION PROGRAM Header
>214D	>0000 Pointer to next Application Program Header ... none here
>214F	>216F Start address for this program (Main entry point)
>2151	>08 Name length for this program
>2152	>54492042 DATA :TI BASIC: (for the menu screen)
>2156	>41534943
>215A	>422B Vector for routine that erases the symbol table (>222B)
>215C	DATA for the cursor character pattern
>2164	DATA for the screen edge character pattern
>216C	DATA for VDP Registers 2, 3 and 4 (>F0 0C F8)
	Note: first nibble is ignored for reg 2 & 4 data.
	register 4 data sets the Pattern Desc. base address to >8*>800 = >4000 which equates to >0000 for 16K
>216F	**START OF TI BASIC INTERPRETER**
	The input line is scanned for the entries at >2214 and branches to them. If not one of these it executes the direct command (ie: CALL CLEAR or PRINT B+C etc.).
>216F	Entry point for 'NEW' routine
>2192	Secondary entry point for Basic Interpreter
>2195	Routine to clear flags, set up keyboard & prepare for input on the command line.
>21D6	Edit Routine that CALLs other routines to store the input from the keyboard into the VDP RAM Screen Image Table.
>21E5	Routine that CALLs another routine to scan the line just input and convert it into token codes and store it in VDP RAM
>2214	CASE branch table for 'RUN NEW CONTINUE LIST BYE NUMBER OLD RESEQUENCE SAVE and EXIT'

TI BASIC INTERPRETER Cont.

>222B	Entry point for routine that erases the Symbol Table
>2245	Entry point for 'LIST' routine
>224D	Entry point for 'RUN' routine
>2268	Entry point for 'CONTINUE' routine
>228C	Entry point for 'NUMBER' routine
>229F	Entry point for 'SAVE' routine
>22A7	Entry point for 'OLD' routine
>22AA	Entry point for 'RESEQUENCE' routine
>2342	Entry point for 'BYE' routine
>236D	Routine to move blocks of VDP RAM from a Lower address to a Higher address as you input program lines. >8300 = VDP location to move FROM >8302 = VDP location to move TO >835C = Number of bytes to move
>2377	Entry point for 'EDIT' (program lines) routine
>2417	Routine to begin execution of Basic program
>2457	Routine to scan an inputted command line CALLed from >21E5
>266C	Routine to generate the SYNTAX ERROR message.
>267E	Routine to restore cursor position after Error
>26AB	Routine to reset the length byte for strings and numerics
>27E3	Routine that clears the screen, resets the cursor and edge characters and then executes the following routine
>27F1	Routine that CALLs routines in GROM 0 to load character sets, then it resets the foreground and background colors and resets VDP Registers 2, 3 and 4

>2828 **VECTOR TABLE FOR EDIT & PRESCAN ROUTINES** (>2000 offset)

>2828	>4FFF	Prescan (builds symbol table and checks for errors)
>282A	>4F43	Generates Bad Line Number error message
>282C	>4C75	Routine to parse the input line for non space chars
>282E	>4DFA	Lists a program line to screen (converts token code into ASCII, reserved words)
>2830	>4CA6	Gets a valid character from the input line
>2832	>4A42	Main edit routine to read in a line from the keyboard
>2834	>4C36	Starts auto number with our line # and increment.
>2836	>4FC4	Finds where the first token is stored in vdp ram for line
>2838	>4BD6	Deletes and inserts program lines (moves memory around)
>283A	>4F12	Checks for valid line number
>283C	>4EF9	Converts a line number from ASCII into Binary value
>283E	>4F5D	Locates a program line in vdp ram
>2840	>4C2B	Starts auto number with default values of 100,10
>2842	>4FAF	Converts line # from Binary to ASCII and displays it
>2844	>5493	Checks for room for symbol table or pab, this routine may execute a garbage collection and try again
>2846	>5450	Checks for type of char 0-9 a-z A-Z etc.
>2848	>51E5	Places a variable in the symbol table
>284A	>522B	Puts dummy entries into the symbol table
>284C	>4D24	Prints out the WARNING messages
>284E	>4D99	Prints out the ERROR messages
>2850	>4C84	Checks the GPL stack and moves a char into it
>2852	>4CA0	Increments the VDP pointer for the next char
>2854	>4CC0	Handles unquoted strings adds unquoted token & len to it
>2856	>4C7A	Gets first non space char from the input line
>2858	>4A49	Secondary edit routine, allows different line length
>285A	>4A4F	Third edit routine, allows different starting cursor pos

>285C	**RESERVED WORD TOKEN TABLE** First 10 words point to the start of reserved word groupings. Groups are broken up by number of characters (1-10) per reserved word. The Token value follows the reserved word
>2A42	**LINE EDITOR**
>2A42	Routine that accepts keystrokes into a screen line. This is a line editor, it knows Insert, Delete etc. This entry point sets the default starting point and line length for Basic
>2A49	Second entry point for the line editor. By setting the line length in >835E before branching here you can change the maximum line length
>2A4F	Third entry point for the line editor. By setting the line length in >835E and the start point in >8361 before branching here you can have your input start and stop any place on the screen
>2BD6	Routine that moves memory around for inserting and deleting program lines
>2C2B	Routine that sets up the values for NUMBER (auto line numbering)
>2C75	Routine that parses a line and gets the non space chars
>2C7A	Routine that gets the first non space char. Both this routine and the one above CALL the routine at >2CA6
>2C84	Routine that checks the stack and moves a char to it
>2CA0	Routine that increments the VDP pointer and jumps to >2C84
>2CA6	Routine that checks for strings or numerics and handles each
>2CC0	Routine that handles unquoted strings, adds token & length to it
>2D24	Routine that prints the WARNING message on the screen
>2D99	Routine that prints the ERROR message on the screen. The pointer to the length byte in GROM for the WARNING or ERROR message is in the Scratch Pad at >8376
>2DFA	Routine that lists a program line on the screen. Starting point for the line is in >8302
>2EF9	Routine that converts an ASCII line number into binary
>2F12	Routine that checks for valid line number input
>2F43	Routine that generates the BAD LINE NUMBER error message
>2F5D	Routine that finds a line from the line number table
>2FAF	Routine that converts line # from binary to ASCII & displays it
>2FC4	Routine that finds the first token of a program line in VDP
>2FFF	PreScan routine, scans line or program and builds symbol table
>31E5	Routine that places the variable in the symbol table
>3450	Routine that checks char type 0-9, a-z, A-Z etc. Character that is checked is at >8342. This routine sets the condition bit in the GPL Status register if char is valid for variable name
>3493	Routine that checks for enough room for a symbol table entry or a PAB. If there's not enough room between the symbol table and the string space it tries to move the string space to a lower address, this may execute a garbage collection. If there still isn't enough room it generates the MEMORY FULL error message. (Word at >834A = space needed in bytes)
	NOTE: Most of the above routines use the FAC and ARG sections of Scratch Pad RAM for parameter passing. Some of them will use the temporary space at >8300 - >8316. Usually whenever a a routine does anything with a single character the character is at >8342. Also, most of the references to Scratch Pad are with an offset of >8300. ie: opcode BF 14 0008 = Double byte store 0008 at >8314

	BRANCH TABLE FOR A FEW OF THE ERROR MESSAGES
>3510	>05 5671 = Branch to 5671 - ILLEGAL STATEMENT
>3513	>05 567D = Branch to 567D - MEMORY FULL
>3516	>05 4D7C = Branch to 4D7C - BAD VALUE
>3519	>05 4D81 = Branch to 4D81 - STRING-NUMBER MISMATCH
	ENTRY POINTS FOR A FEW OF THE CALL STATEMENTS
>351C	**CLEAR** - Places the space character + bias (>60) in every screen position by using the GPL statement of ALL : :
>3527	DATA for SOUND >42,>0B,>12,>22,>00,>00,>00,>00
>352F	DATA for SOUND >01,>FF,>01,>04,>9F,>BF,>DF,>FF,>00
>3538	**SOUND** - This routine handles the entire sound statement. First it checks the duration, then it converts it into 1/60 seconds because sounds are interrupt driven. Next it finds the first frequency and divides 111834 by it (111834/freq) and passes that value to a sound table it is setting up in VDP RAM. Next it gets the volume and sets that up and then passes all the values to the sound chip (>8400). Interrupt routine is in the console ROM chip.
>360E	**HCHAR** - This routine and the VCHAR routine both call a subroutine at >37D6 to parse the statement for X,Y,CHAR,#CHRS and converts these into integer values. Then it puts them on the screen using a FMT statement (Formatted block move) that allows for writing over the border characters.
>362A	**VCHAR** - This is very similar to the above statement except that it places the characters vertically. The number of characters is at >834A, the character is at >8300, screen row is at >837E and the screen column is at >837F.
>3643	**CHAR** - This routine converts the string into the proper values for defining a character and moves these values into VDP RAM at the proper character + bias (>60) location. Both FAC (>834A) and ARG (>835C) are heavily used during this CALL. This routine appears to set up a temporary string in VDP RAM so it is possible that it could invoke a garbage collection and if there isn't enough room it will generate a Memory Full error message.
>3708	**KEY** - This parses the statement for the key unit, checks it for the proper range, CALLs >3767 to move it to >8374 and then executes the SCAN routine. After returning it checks the Status and places the proper value into your variable. Next it evaluates the keycode, converts it into floating point and places it in your variable.
>3748	**JOYST** - This is very similar to the above statement except after returning from >3767 it computes the proper X and Y values by CALLing >5755 and then places them into your variables.
>3767	Subprograms to do parsing for left parenthesis and commas, range checking for a range of 1-16, >0 or a preselected range.
>378E	Subprogram to parse the row and column values out of a graphics statement (ie: CALL HCHAR...).
>37BF	**SCREEN** - This subprogram sets the Screen and border color. It uses the above subroutines to parse the statement and then places the value into VDP register 7.
>37D6	Subprogram to parse HCHAR and VCHAR statements for row, column (by CALLing >378E), ASCII character value and number of characters.

>4000	**VECTOR TABLE FOR FILE ROUTINES** (>0000 Offset)
>4000	>426C DISPLAY routine
>4002	>4160 DELETE routine
>4004	>4227 PRINT routine
>4006	>4344 INPUT routine
>4008	>401E OPEN routine
>400A	>4174 CLOSE routine
>400C	>41D7 RESTORE routine
>400E	>45E3 READ routine
>4010	>4956 GET DATA FROM GROM/GRAM or RAM
>4012	>41CF CLOSE ALL OPEN FILES routine
>4014	>46FC PROGRAM SAVE routine
>4016	>4641 PROGRAM LOAD routine
>4018	>474C LIST routine
>401A	>4BFC OUTPUT RECORD routine
>401C	>482B END OF FILE routine
>401E	**OPEN ROUTINE** - This handles OPEN #x:"device.xx",VARIABLE xx,...
>40AF	Case branch table for the following OPEN parameters:
>4051	>40AB VARIABLE
>4053	>406B RELATIVE
>4055	>40D1 INTERNAL
>4057	>4070 SEQUENTIAL
>4059	>4095 OUTPUT
>405B	>409A UPDATE
>405D	>40A4 APPEND
>405F	>40B0 FIXED
>4160	**DELETE ROUTINE** - This handles the various DELETE functions
>4174	**CLOSE ROUTINE** - This handles CLOSE #x or CLOSE #x:DELETE
>41CF	**CLOSE ALL FILES ROUTINE** - This closes all open files
>41D7	**RESTORE ROUTINE** - This handles RESTORE (data), RESTORE xx (data) RESTORE #x and RESTORE #x,REC x for files
>4227	**PRINT ROUTINE** - This handles both screen and file PRINT. Both this and the Display routine check for Internal or Display type records and handle each accordingly.
>426C	**DISPLAY** - This handles the screen DISPLAY statement (no files)
>4344	**INPUT ROUTINE** - This handles both the screen and file INPUT it also checks data type against variable type
>45E3	**READ ROUTINE** - This handles the reading of DATA into variables it is not used for files. CALL's routines at >48CC - >4992
>4641	**OLD ROUTINE** - This is the OLD DSK1.xxxxx or OLD CS1 routine, it sets up the PAB, Calls the DSR, Tests the Checksum, gets the new addresses for the end & start of the line # table, makes adjustments for different RAM size (4K?) and stores them at >8332 & >8330 respectively. Adjusts the memory and updates the line # pointers if different RAM size. Both OLD & SAVE CALL routines at >4888 - >48CB
>46FC	**SAVE ROUTINE** - This is SAVE DSK1.xxx or SAVE CS1, it closes all open files, clears all break points, stores the start and end pointers for the line # table, finds the number of bytes used (>8370), passes it to the PAB and calls the DSR for a SAVE.
>474C	**LIST ROUTINE** - This lists out the program lines to the screen or to the device specified. Unfortunately it generates a Syntax error if you use anything but a : after the device name. ie; LIST "PIO":100-150 is OK but not "PIO",VARIABLE 28
>482B	**END OF FILE ROUTINE** - This is the EOF(x) function.

	SUBROUTINES
>4888	**OLD & SAVE SUBROUTINE** - This gets the program name, initializes many of the program pointers, deletes the symbol table, sets up the PAB and returns.
>48CC	**READ & INPUT SUBROUTINES** - These find the symbol table entries, check for Strings or Numerics, decide if its GROM or RAM data and pass the Data item to the variable.
>4956	- GET DATA FROM GROM OR RAM - Reads the next Data item from GROM if the GROM Flag at >8389 is in >834D. If >834D is = 0 then the next Data item is read from RAM.
>4993	**OPEN, CLOSE & RESTORE SUBROUTINES** - These parse out the file number (ie: #1, if its there), check for the proper range (> 0 and < 256), scan the PAB chain for the proper file. If any of these items are not right it returns with an error. On a Close the routine at >49E6 deletes the PAB and adjusts the memory and PAB chain pointers.
>4B53	**PRINT SUBROUTINES** - These handle the outputting of data to the screen or to a file. They check for valid separators (,;:) and handle each accordingly. For screen output they add the character offset (>60) to each character.
>4BFC	- OUTPUT A RECORD - This is the subroutine that outputs a a record to either the screen or an output device, depending on the PAB (file #0 = screen output)
>4D00	**VECTOR TABLE FOR BASIC EXECUTION**
>4D00	>56CD Screen Scroll Routine
>4D02	>5120 Move a String from the Program area to the String Space
>4D04	>4DB0 Second entry point for executing a Basic Program
>4D06	>56BB Subroutine to find line number after BREAK
>4D08	>5613 Subroutine that sets the pointer for next Data item
>4D0A	>5645 Subroutine to convert line number into ASCII (Trace mode)
>4D0C	>4DBF Third entry point for executing a Basic program (CONT)
>4D0E	>4E38 Subroutine that BREAKs a running program
>4D10	>4D8A First entry point for executing a Basic program (RUN)
>4D12	>515C Subroutine that sets up room for a String
>4D14	>55BB Subroutine that clears out a temporary String
>4D16	>56E1 Subroutine to convert a String into a Number
>4D18	>51A9 Garbage Collection subroutine.
>4D1A	**SUBPROGRAM POINTER TABLE** (For CALL xxxx...)
>4D1A	>4D24 Points to next Subprogram
>4D1C	>3538 Entry point for this Subprogram
>4D1E	>05 Length of this name
>4D1F	>534F554E44 :**SOUND:**
>4D24	>4D2E Points to next Subprogram
>4D26	>351C Entry point for this Subprogram
>4D28	>05 Length of this name
>4D29	>434C454152 :**CLEAR:**
>4D2E	>4D38 Points to next Subprogram
>4D30	>5713 Entry point for this Subprogram
>4D32	>05 Length of this name
>4D33	>434F4C4F52 :**COLOR:**
>4D38	>4D42 Points to next Subprogram
>4D3A	>56EF Entry point for this Subprogram
>4D3C	>05 Length of this name
>4D3D	>4743484152 :**GCHAR:**

	SUBPROGRAM POINTER TABLE Cont.
>4D42	>4D4C Points to next Subprogram
>4D44	>360E Entry point for this Subprogram
>4D46	>05 Length of this name
>4D47	>4843484152 :**HCHAR:**
>4D4C	>4D56 Points to next Subprogram
>4D4E	>362A Entry point for this Subprogram
>4D50	>05 Length of this name
>4D51	>5643484152 :**VCHAR:**
>4D56	>4D5F Points to next Subprogram
>4D58	>3643 Entry point for this Subprogram
>4D5A	>04 Length of this name
>4D5B	>43484152 :**CHAR:**
>4D5F	>4D67 Points to next Subprogram
>4D61	>3708 Entry point for this Subprogram
>4D63	>03 Length of this name
>4D64	>4B4559 :**KEY:**
>4D67	>4D71 Points to next Subprogram
>4D69	>3748 Entry point for this Subprogram
>4D6B	>05 Length of this name
>4D6C	>4A4F595354 :**JOYST:**
>4D71	>0000 Points to next Subprogram (no more)
>4D73	>37BF Entry point for this Subprogram
>4D75	>06 Length of this name
>4D76	>53435245454E :**SCREEN:**
>4D7C	Generate 'BAD VALUE' Error Message
>4D81	Generate 'STRING-NUMBER MISMATCH' Error Message
>4D86	>56D4 - Branch to routine that sets up format for screen
>4D88	>566C - Branch to CAN'T DO THAT Error
>4D8A	**RUN** - This is where a Basic program first starts to RUN. This sets up the line number pointers, scrolls the screen up 1 line and falls through to the next entry.
>4DB0	**EXECUTE** - This starts execution of the program or if in Command mode it executes the statement you just typed in.
>4DBF	Third Entry point for Basic program execution. This is where the CONTINUE Command branches to.
>4E38	Subroutine that BREAKs a running program. It prevents a break while GROM is executing, sets up the BREAK message and displays the line number.
>4E5B	**** DONE **** - This is the normal end of program subroutine.
	VECTOR TABLE FOR BASIC RESERVED WORDS
>4E84	>4FB6 FOR
>4E86	>5463 BREAK
>4E88	>5479 UNBREAK
>4E8A	>5459 TRACE
>4E8C	>545E UNTRACE
>4E8E	>400E READ
>4E90	>4004 PRINT
>4E92	>50DB CALL
>4E94	>5111 QUOTED STRING CONSTANT
>4E96	>400C RESTORE
>4E98	>50CB RANDOMIZE
>4E9A	>4006 INPUT
>4E9C	>4008 OPEN
>4E9E	>400A CLOSE
>4EA0	>4F99 ((Left Parenthesis)

	VECTOR TABLE FOR BASIC RESERVED WORDS Cont.
>4EA2	>4FB2 + (Plus)
>4EA4	>4FA8 - (Minus)
>4EA6	>4ED1 ABS
>4EA8	>4EDC ATN
>4EAA	>4EE2 COS
>4EAC	>4EE8 EXP
>4EAE	>4EEE INT
>4EB0	>4EFA LOG
>4EB2	>4F26 SGN
>4EB4	>4F40 SIN
>4EB6	>4F46 SQR
>4EB8	>4F4C TAN
>4EBA	>52BE LEN
>4EBC	>53EA CHR$
>4EBE	>4F00 RND
>4EC0	>4000 DISPLAY
>4EC2	>4002 DELETE
>4EC4	>524A SEG$
>4EC6	>531A STR$
>4EC8	>5349 VAL
>4ECA	>53A9 POS
>4ECC	>5306 ASC
>4ECE	>05 401C EOF
NOTE-------	Rather than document each of the above items, which would require another 4-6 pages of memory maps, we will talk about these routines in general. First off, many of these routines end with the Opcode of >10 this is the same as Basic's CONT, so the interpreter will go back to >4DBF and grab the next statement in your Basic Program. All of these routines use various parts of Scratch Pad RAM with FAC (>834A) and ARG (835C) being used very heavily. There is also a 24 byte segment at the top of Scratch Pad RAM (>8300 through >8316) used by Basic as temporary storage places for many of its routines. Some of the routines will clear out any values it has place into the FAC and ARG area or the Row, Column and Character value area at >837D - >837F. Most of the String handling routines require that FAC through FAC + 7 (>834A - >8351) be set up prior to execution as follows: >834A = The Symbol table address that points to the string. >834C = >6500 for a string and >6400 for numerics. >834E = The address in VDP RAM of the string. >8350 = The length of the string. *>8352 - Sometimes the GROM Flag is temporarily stored here
>54CF	Subroutine to handle User Defined Functions (ie: DEF)
>5600	Subroutine to check for String or numeric and set register bits.
>5613	Subroutine to set the pointer for DATA items.
>5645	Subroutine to convert the Line number into ASCII.
>565C	Subroutine to print out an Error Message.
>56BB	Subroutine to find line # after BREAK, UNBREAK or RESTORE.
>56EF	**GCHAR** subroutine.
>5713	**COLOR** subroutine.
>5740	Subroutine to convert floating point to integer.
>5755	Subroutines used by CALL JOYST and CALL KEY.
>57AB	Subroutine to check for the left parenthesis (.
>57C0	Error Message subroutines.

5TH 1- =FORTH

This month we have a little Forth routine that turns on the Peripheral DSR ROMs and searches through them for the various headers. Each peripheral determines its own headers, so all the headers will not be in every ROM. The valid list of headers are as follows:

>4000 = Header Identification & Version
>4002 = # of Programs (not used in DSRs)
>4004 = Power Up Header Address
>4006 = User Program Header (not used)
>4008 = DSR Link Header Address
>400A = Subprogram Header Address
>400C = Interrupt Header Address
>400E = Reserved (not used)

On a two or more disk drive system just type in the program exactly as it is printed and then LOAD the screen you saved it to.

On a one drive system this program it will try to load the -CRU words from the Forth system disk. If you haven't saved this program on your system disk then delete -CRU from line 0 Screen 110 and type in -CRU <enter> before LOADing the screen with the program on it.

--------------- WARNING -----------------

If you use the any of the following words: ON, DISKON, RS232ON, TPON or PCODEON be sure to type in ALLOFF when you are finished. If you do not and you leave a DSR ROM ON you will lock up your system when you go to access ANY of the peripherals!!!

This happens because two DSRs can not occupy the DSR ROM space at the same time (do not use SWCH for the printer).

The words DISKON, RS232ON, TPON and PCODEON turn on these DSRs. When some of them are on, like the Disk, their light will also come on. However, some of them, like the RS232, will not turn on the light. After a DSR is turned on you can DUMP its contents to the screen with the DUMPDSR word.

The two main words for this program are SEARCH and SHOWALL. These words are just typed in, nothing needs to be on the stack for them to execute. If you just want to see a selected DSR header type in:

HEX cru base HEAD <enter>

ie: HEX 1100 HEAD displays the Disk DSR header.

ALLOFF loops through all of the CRU bases and turns off all of the DSR ROMs. Make sure they are all off when you are finished!!

Quite a few of our readers have written asking how to convert their Forth system disks into Double sided and/or Double density.

CorComp recently sent out a newsletter called the CorComp Cursor that contained a two page article on converting your Forth systems disks that was written by Jim Vincent. This article covered all of the changes you need to make to the screens that are disk size dependent including the changes to the FORMAT-DISK word. Rather than use up space here to repeat Jims instructions try to get a copy of this newsletter from your Group. If you can not get this info AND if we get a lot of requests for it we will publish it in a future issue. Have Fun.

SCR #110

```
 0 ( Peripheral DSR Peeker - Craig Miller) BASE->R HEX     -CRU
 1 : OFF       2 / SBZ ;                ( Turns OFF selected DSR ROM )
 2 : ALLOFF    2000 1000 DO I OFF 100 +LOOP ; ( Turns OFF all DSRs)
 3 : ON        ALLOFF 2 / SBO DECIMAL ;   ( Turns ON selected DSR )
 4 : DISKON    1100 ON ;               ( TI Disk Controller Card   ON )
 5 : RS232ON   1300 ON ;               ( TI RS232 Interface Card   ON )
 6 : TPON      1800 ON ;               ( TI Thermal Printer DSR    ON )
 7 : PCODEON   1F00 ON ;               ( TI P-Code Card            ON )
 8 : DUMPDSR   4000 1FFF DUMP ; ( Dumps Enabled DSR ROM to screen )
 9 : SEARCH    PAGE ." Peripheral DSRs found at :" CR CR
10             ." CRU Base   Version" CR CR ALLOFF HEX
11             2000 1000 DO I DUP 2 / SBO 4000 C@ AA = IF I 5 .R
12             4001 C@ 9 .R CR ENDIF OFF 100 +LOOP CR DECIMAL
13             ." To enable DSR ROM type in" CR
14             ." HEX  cru base  ON  <Enter>  " ;
15 -->         ( Note: PAGE = 0 0 GOTOXY CLS from Screen 3 )
```

SCR #111

```
( Peripheral DSR Peeker cont.)
: P         5 U.R ;
: PR        DUP P @ P ;  ( HEX cru base HEAD lists DSR header )
: HEAD1     DUP 2 / SBO  HEX 4000 C@ AA =        IF CR CR
            4000 PR ."   Header Ident & Version"    CR
            4002 PR ."   # of Programs (not used)"  CR
            4004 PR ."   Power Up Header Address"   CR
            4006 PR ."   User Program Header n/u"   CR
            4008 PR ."   DSR Header Address"        CR
            400A PR ."   Subprogram Header Address" CR
            400C PR ."   Interrupt Header Address"  CR
            400E PR ."   Reserved not used"         CR
               ELSE ."  - No DSR here"
               ENDIF CR ;
: HEAD      CR ALLOFF HEAD1 OFF DECIMAL ;
-->
```

SCR #112

```
( Peripheral DSR Peeker cont.)
: ?PAUS     PAUSE IF ALLOFF DECIMAL SP! QUIT ENDIF ;
: FINI      ?PAUS @ 0= DUP IF SWAP DROP ENDIF ;
: EMITASC   DUP DUP 19 > SWAP 7F < AND IF EMIT ELSE . ENDIF ;
: ?HEAD     DUP @ 0 > DUP 0= IF SWAP DROP ." none" ENDIF CR CR ;
: 3DUP      @ DUP DUP DUP ;
: NAMES     CR 4  + DUP DUP DUP P C@ P ."   Name Length" CR
            C@ 1+ + SWAP 5 + ."       name=  "
            DO I C@ EMITASC LOOP CR CR FINI ;
: PWUHDS    4004 ?HEAD IF BEGIN 3DUP                   PR
            ."   Next Power Up Header"    CR    2+ PR
            ."   Entry for this Power Up" CR  FINI UNTIL ENDIF ;
: DSRHDS    4008 ?HEAD IF BEGIN 3DUP DUP DUP           PR
            ."   Next DSR Link Header"    CR    2+ PR
            ."   Entry for this DSR Link"    NAMES UNTIL ENDIF ;
-->
```

SCR #113

```
( Peripheral DSR Peeker cont.)
: SUBHDS    400A ?HEAD IF BEGIN  3DUP DUP DUP          PR
            ."   Next Subprogram Header"   CR    2+ PR
            ."   Entry for this Subprogram"  NAMES UNTIL ENDIF ;
: INTHDS    400C ?HEAD IF BEGIN  3DUP                  PR
            ."   Next Interrupt Header"    CR    2+ PR
            ."   Entry for this Interrupt" CR FINI UNTIL ENDIF ;

: SHOWALL   PAGE ALLOFF HEX 2000 1000 DO CR I DUP U.
            ." DSR Header " HEAD1 ?PAUS 4000 C@ IF
            ." Power Up Header "   PWUHDS CR
            ." DSR Link Header "   DSRHDS CR
            ." Subprogram Header " SUBHDS CR
            ." Interrupt Header "  INTHDS CR CR ENDIF
            OFF ?PAUS 100 +LOOP DECIMAL ;
R->BASE
```

S U B S C R I P T I O N   I N F O R M A T I O N

THE SMART PROGRAMMER - a monthly 16+ page newsletter published by **MILLERS GRAPHICS**
U.S. 12.50 year - Foreign Surface Mail 16.00 year - Foreign Air Mail 26.00 year

Back issues are available. We can start your subscription with the FEB. 84 issue
To subscribe send a Check, Money Order or Cashiers Check, payable in U.S. currency

TO: **MILLERS GRAPHICS**
1475 W. Cypress Ave.
San Dimas, CA 91773

MG

MILLERS GRAPHICS
1475 W. Cypress Ave.
San Dimas, CA 91773

BULK RATE
U.S. POSTAGE
PAID
San Dimas, CA 91773
PERMIT NO. 191

THE SMART PROGRAMMER